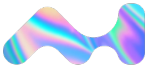


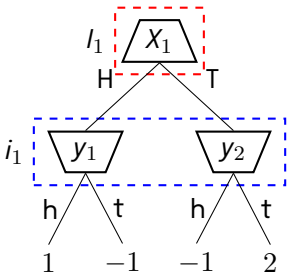
Solving Imperfect Information EFGs

Ondřej Kubíček

Artificial Intelligence Center
Faculty of Electrical Engineering
Czech Technical University in Prague

October 24, 2023

Problems with solving imperfect information games

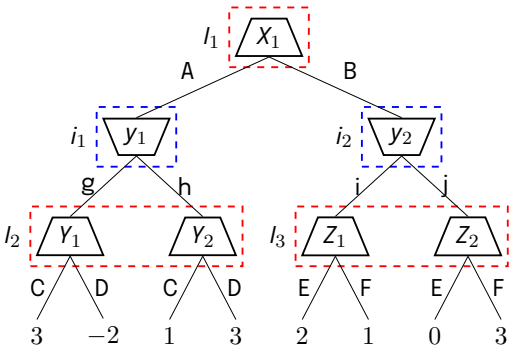


- First player in I_1 chooses between H and T based on the strategy that the player 2 plays.
- Second player in i_1 chooses between h and t based on the probability if it is in y_1 or y_2, that directly corresponds to the strategy of player 1.
- Backward induction will not work because of this interconnected dependency.
- Generally even in perfect recall imperfect information games, the policy depends on both policy in previous and subsequent parts of the game tree

Example

AI CENTER FEE CTU

- Induced normal-form game can include the same leaf multiple times.
- Playing some actions may invalidate different actions in future.

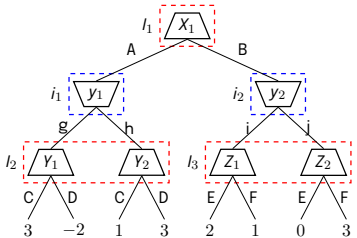


	gi	gj	hi	hj
ACE	3	3	1	1
ACF	3	3	1	1
ADE	-2	-2	3	3
ADF	-2	-2	3	3
BCE	2	0	2	0
BCF	1	3	1	3
BDE	2	0	2	0
BDF	1	3	1	3

Sequences

AI CENTER FEE CTU

- Ordered list of all the actions that may be played in a single playthrough of the game is called Sequence.
- We denote all possible sequences of player *i* as Σ_i.



Σ_1	Σ_2
$\emptyset$	$\emptyset$
A	g
B	h
AC	i
AD	j
BE	
BF	

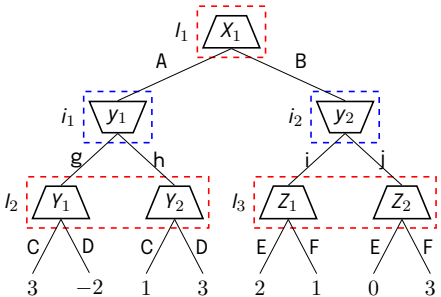
Realization plans

- Realization plan $r_i(\sigma_i)$ is a probability that σ_i will be played, assuming that the other player plays only actions that allow σ_i to be executed
- Let us assume that σ_i leads to the infoset I_i. Behavioral strategy $\pi(I_i, a)$ of playing action a in this infoset is computed as

$$\pi(I_i, a) = \begin{cases} \frac{r(\sigma_i a)}{r(\sigma_i)} & \text{if } r(\sigma_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\sigma_i a$ represents a extensions of sequence σ_i with action a.

Example

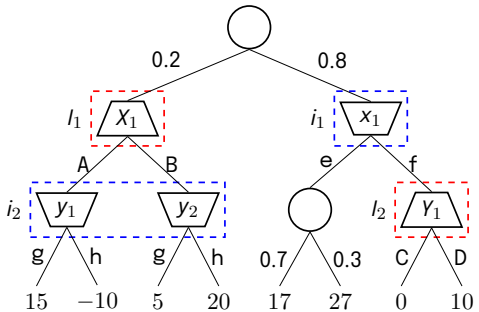


Σ_1	Σ_2
$\emptyset$	$\emptyset$
A	g
B	h
AC	i
AD	j
BE	
BF	

- $r_1(\emptyset) = 1$
- $r_1(\mathsf{A}) + r_1(\mathsf{B}) = r_1(\emptyset)$
- $r_1(\mathsf{AC}) + r_1(\mathsf{AD}) = r_1(\mathsf{A})$
- $r_1(\mathsf{BE}) + r_1(\mathsf{BF}) = r_1(\mathsf{B})$

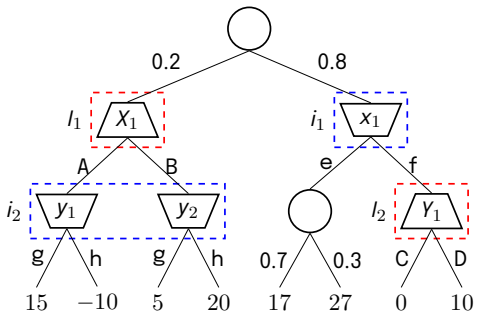
- $r_2(\emptyset) = 1$
- $r_2(\mathsf{g}) + r_2(\mathsf{h}) = r_2(\emptyset)$
- $r_2(\mathsf{i}) + r_2(\mathsf{j}) = r_2(\emptyset)$

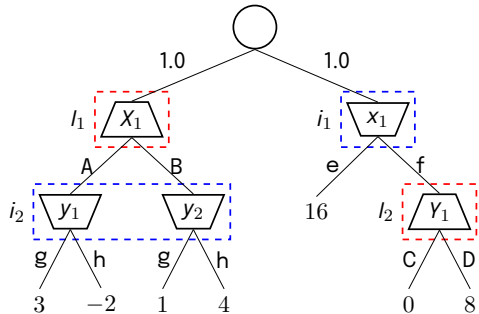
Propagating chance node



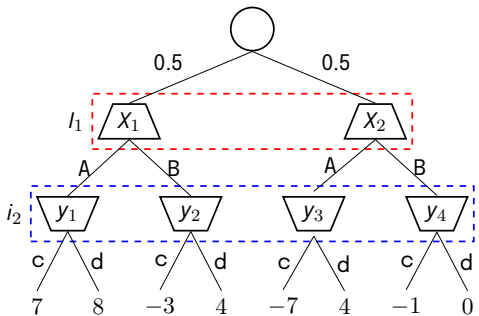
- Chance nodes have fixed probabilities in the game
- These probabilities can be propagated from the chance node to the terminal utilities
- Chance nodes that do not reveal any information until the end of the game can be prunned away completely

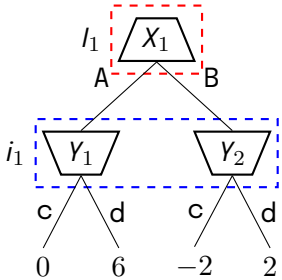
Example


0.2
0.8
I_1
X_1
i_1
x_1
A
B
e
f
i_2
y_1
y_2
I_2
Y_1
g
h
g
h
0.7
0.3
C
D
15
−10
5
20
17
27
0
10


1.0
1.0
I_1
X_1
i_1
x_1
A
B
e
f
i_2
y_1
y_2
16
I_2
Y_1
g
h
g
h
C
D
3
−2
1
4
0
8

Complete removal of chance node





Extended utility function

AI CENTER FEE CTU

- Extended utility function for sequences is $g : \Sigma_1 \times \Sigma_2 \to \mathbb{R}$

$$g(\sigma_1, \sigma_2) = \sum_{z \in \mathcal{Z}'} \mathcal{C}(z) u(z)$$

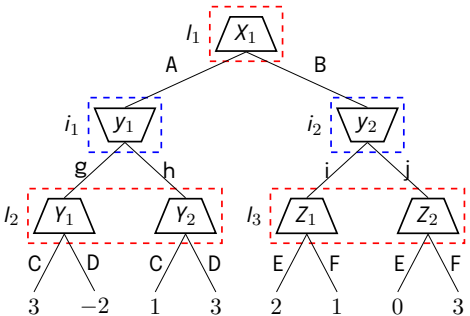
- $\mathcal{C}(z)$ is a probability that leaf z was reached due to chance nodes along the way.
- $\mathcal{Z}' \subseteq \mathcal{Z}$ are all the terminal histories that could be reached by sequences σ_1, σ_2

Example

$g(\emptyset, \emptyset) = 0$
$g(\mathsf{AC}, \mathsf{i}) = 0$

$g(\mathsf{BF}, \mathsf{j}) = 3$
$g(\mathsf{AD}, \emptyset) = 0$

$g(\emptyset, \mathsf{g}) = 0$
$g(\mathsf{A}, \mathsf{g}) = 0$



Σ_1	Σ_2
$\emptyset$	$\emptyset$
A	g
B	h
AC	i
AD	j
BE	
BF	

Sequence-form Linear Program (SQF)

AI CENTER FEE CTU

$$\max_{r_1, v} v(I_{\text{root}}) \tag{1}$$

$$\text{s.t.} \quad r_1(\emptyset) = 1 \tag{2}$$

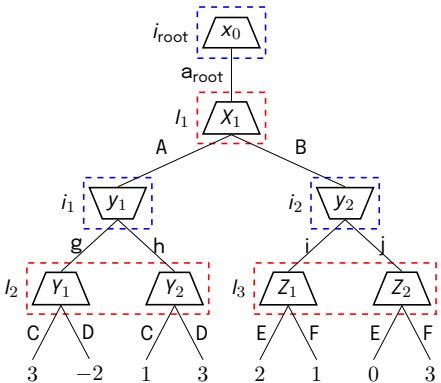
$$r_1(\sigma_1) \geq 0 \qquad \forall \sigma_1 \in \Sigma_1 \tag{3}$$

$$\sum_{a \in \mathcal{A}_1(I_1)} r(\sigma_1 a) = r_1(\sigma_1) \qquad \forall I_1 \in \mathcal{I}_1, \sigma_1 = \varsigma_1(I_1) \tag{4}$$

$$\sum_{I_2' \in \mathcal{I}_2 : \sigma_2 a = \varsigma_2(I_2')} v(I_2') + \sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r_1(\sigma_1) \geq v(I_2) \quad \forall I_2 \in \mathcal{I}_2, \sigma_2 = \varsigma_2(I_2), \forall a \in \mathcal{A}(I_2) \tag{5}$$

- Variables are realization plans r_1 for all the sequences of player 1 and expected values v for each opponents infoset, if it plays a best response.
- ς_i returns for a given infoset I_i a sequence σ_i that leads to this infoset.

Example



$$\max_{r_1, v}(v(i_{\text{root}}))$$

$$r_1(\emptyset) = 1$$

$$r_1(\mathsf{A}) + r_1(\mathsf{B}) = r_1(\emptyset)$$

$$r_1(\mathsf{AC}) + r_1(\mathsf{AD}) = r_1(\mathsf{A})$$

$$r_1(\mathsf{BE}) + r_1(\mathsf{BF}) = r_1(\mathsf{B})$$

$$v(i_1) + v(i_2) \geq v(i_{\text{root}})$$

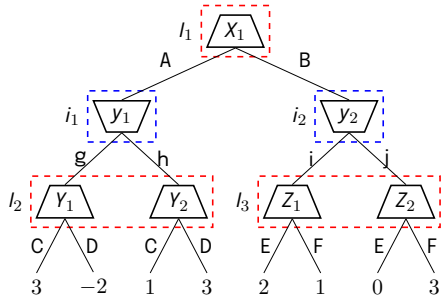
$$3r_1(\mathsf{AC}) - 2r_1(\mathsf{AD}) \geq v(i_1)$$

$$1r_1(\mathsf{AC}) + 3r_1(\mathsf{AD}) \geq v(i_1)$$

$$2r_1(\mathsf{BE}) + 1r_1(\mathsf{BF}) \geq v(i_2)$$

$$0r_1(\mathsf{BE}) + 3r_1(\mathsf{BF}) \geq v(i_2)$$

Example



$$\min_{r_2, v}(v(I_1))$$

$$r_2(\emptyset) = 1$$

$$r_2(\mathsf{g}) + r_2(\mathsf{h}) = r_2(\emptyset)$$

$$r_2(\mathsf{i}) + r_2(\mathsf{j}) = r_2(\emptyset)$$

$$v(I_2) \leq v(I_1)$$

$$v(I_3) \leq v(I_1)$$

$$3r_2(\mathsf{g}) + 1r_2(\mathsf{h}) \leq v(I_2)$$

$$-2r_2(\mathsf{g}) + 3r_2(\mathsf{h}) \leq v(I_2)$$

$$2r_2(\mathsf{i}) + 0r_2(\mathsf{j}) \leq v(I_3)$$

$$1r_2(\mathsf{i}) + 3r_2(\mathsf{j}) \leq v(I_3)$$

Sequence-form properties

AI CENTER FEE CTU

- The fastest exact algorithm
- Easy to implement
- Poor scaling due to memory requirements of the linear program
- Hard to fine-tune for specific domains to increase performance
- Cannot be used to solve the game only partially

Double Oracle Algorithm

AI CENTER FEE CTU

- Large linear programs are often solved in a way that some constraints are removed from the LP and this smaller LP is solved.
- If this solution does not violate any removed constraint, than it is a solution the problem.
- Otherwise some of the violated constraints are added to the LP and it is solved again.
- These are called lazy constraints.
- Game Theory often uses similar idea with the name of double oracle.
- The goal of double oracle algorithm is to remove some strategies from the game and solve this smaller game.
- Then it is checked whether this solution is a Nash equilibrium in the original game.

Double Oracle Algorithm

AI CENTER FEE CTU

- The main loop of the double oracle algorithm is
  1. Create a restricted game, that does not contain all the strategies
  2. Solve this restricted game
  3. Compute a best response by each player in the original game
  4. If best responses are the same as the solution of the restricted game, then we have found the Nash equilibrium, otherwise we add these best responses to the restricted game
- In the worst case scenario the double oracle algorithm has to add all the strategies from the original game.
- However, in most cases, the restricted game is much smaller than the original game.
- This algorithm can be applied to both Normal-form and Extensive-form games.

Double Oracle Algorithm with sequences

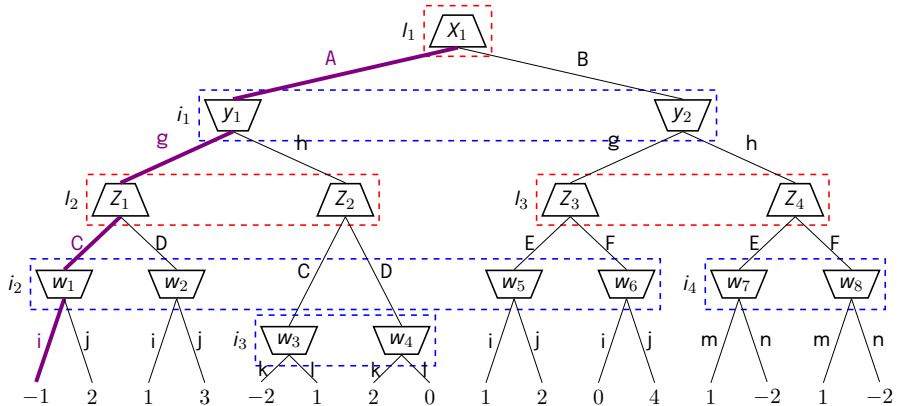
AI CENTER FEE CTU

- Using sequences instead of pure strategies is more complicated.
- With limited amount of sequences, some reachable parts of the game tree may not have any sequence to be played.
- To avoid this, in each information set, there is some default action that should be played.
- Default action does not have to be defined explicitly.
- Instead of keeping the full tree, the nods with default action can be replaced by terminal node.
- The value of this terminal node corresponds to the value if opponent picks best response against the default action.

Example

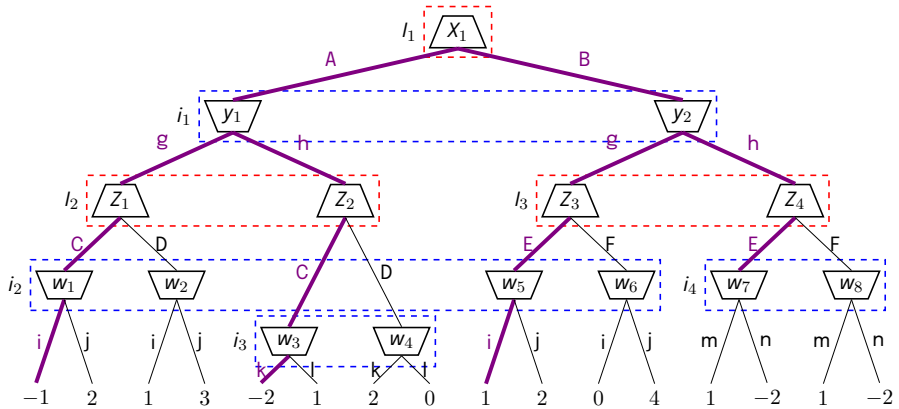
AI CENTER FEE CTU

Let us start with restricted game that contains only sequences AC and gi. Trivially solving this restricted game chooses only the available sequences, with the resulting value of the game being -1.



Example

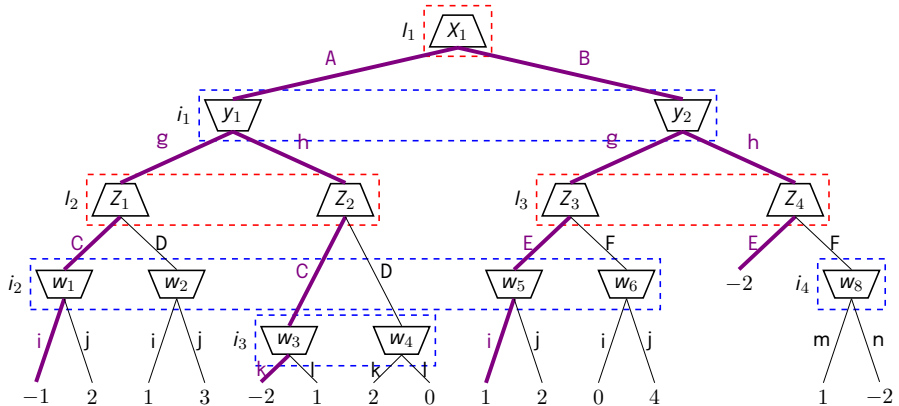
Best response of player 1 is a sequence BE. Best response of player 2 is a sequence hk.



Example

AI CENTER FEE CTU

Sequence BhE does not have any action defined, so the restricted game has to transform it into leaf and assign utility to it. Best response of the player is playing n, so the value assigned is -2. This restricted game may again be solved.



Double Oracle properties

AI CENTER FEE CTU

- Can solve larger games than SQF.
- Without any additional information the algorithm identifies, which strategies are not important for good solution.
- Computing best response is faster than solving the Linear program and can be improved with some heuristics for specific problems.
- Harder to implement, due to the need to construct the valid restricted game.
- Still requires SQF to solve the restricted game, which is the primary limitation of the method.
- In games where all sequences have to be considered, it is slower than the SQF.

Summary

AI CENTER FEE CTU

- Mixed strategies are not well suited for solving the imperfect information EFGs.
- Sequences can be used only in perfect recall games, but avoid necessity to define each action for each information set.
- Probability from chance nodes can be propagated up to the leaf utilities.
- If chance node does not reveal any information until the end of the game, it can be removed from the game completely.
- Sequence-form linear program for solving EFGs expands the ideas from linear program for normal-form games to the extensive-form setting.
- It uses the realization plans instead of mixed strategies and expected values for each opponent's infoset.
- Double oracle algorithm iteratively finds sequences that are important in a game to compute Nash equilibrium.